**FUNDAMENTALS OF PYTHON PROGRAMMING – SET 1 QP**

**PART – A: (Multiple Choice Questions) (1 x 10=10 Marks)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Q.1. Answer ***ALL*** questions | | | [CO#] | [PO#] |
| a. | In which language is Python written? | | **(iii)** |  |
|  | 1. English | 1. PHP |  |  |
|  | 1. C | 1. All the above |  |  |
| b. | Which one of the following is the correct extension of the Python file? | | **(i)** |  |
|  | 1. .py | 1. .python |  |  |
|  | 1. .p | 1. None of the above |  |  |
| c. | Which of the following declarations is incorrect in python language? | | **(ii)** |  |
|  | (i)xyzp = 5,000,000 | (ii)x y z p = 5000 6000 7000 8000 |  |  |
|  | (iii)x,y,z,p = 5000, 6000, 7000, 8000 | (iv) x\_y\_z\_p = 5,000,000 |  |  |
| d. | Which of the following operators is the correct option for power(ab)? | |  |  |
|  | 1. a ^ b | 1. a\*\*b | **(ii)** |  |
|  | 1. a ^ ^ b | 1. a ^ \* b |  |  |
| e. | Study the following statement:  **>>>**"a"+"bc"  What will be the output of this statement? | | **(ii)** |  |
|  | 1. a+bc | 1. abc |  |  |
|  | 1. a bc | 1. a |  |  |
| f. | Command to make thread sleep? | | **(i)** |  |
|  | 1. Thread.Sleep | 1. Thread\_Sleep |  |  |
|  | 1. ThreadSleep | 1. Threadsleep |  |  |
| g. | What does the <readlines()> method returns? | | **(ii)** |  |
|  | 1. Str | 1. A list of lines |  |  |
|  | 1. A list of single characters | 1. List of integers |  |  |
| h. | **When will the else part of try-except-else be executed?** | | **(iii)** |  |
|  | 1. always | 1. when an exception occurs |  |  |
|  | 1. when no exception occurs | 1. when an exception occurs in to except block |  |  |
| i. | [Which keyword is used to fetch the data from the table in database ?](http://r4r.in/mcqs/mcqs-questions-answers.php?que_id=279&test_id=7) | | **(i)** |  |
|  | 1. select | 1. fetch |  |  |
|  | 1. raw | 1. all the above |  |  |
| j. | Expand CGI | | **(iii)** |  |
|  | 1. Computer General Interface | (ii) Common General Interface |  |  |
|  | (iii)Common Gateway Interface | (iv) Computer Gateway Interface |  |  |

**PART – B: (Short Answer Questions) (2 Marks)**

|  |  |  |  |
| --- | --- | --- | --- |
| Q.2. Answer ***ALL*** questions | | [CO#] | [PO#] |
| a. | Differentiate Script mode and interactive mode?  **Script Mode** is used when the user is working with more than one single code or a block of code. It takes more time to compile.  **Interactive mode** is used when an user wants to run one single line or one block of code. It runs very quickly and gives the output instantly |  |  |
| b. | Define any two standard data types with example  Python has five standard Data Types:   * [Numbers](https://developer.rhino3d.com/guides/rhinopython/python-datatypes/#numbers) * [String](https://developer.rhino3d.com/guides/rhinopython/python-datatypes/#string) * [List](https://developer.rhino3d.com/guides/rhinopython/python-datatypes/#list) * [Tuple](https://developer.rhino3d.com/guides/rhinopython/python-datatypes/#tuple) * [Dictionary](https://developer.rhino3d.com/guides/rhinopython/python-datatypes/#dictionary) * Numbers * Python numbers variables are created by the standard Python method: * var = 382 * Most of the time using the standard Python number type is fine. Python will automatically convert a number from one type to another if it needs. But, under certain circumstances that a specific number type is needed (ie. complex, hexidecimal)   String  Create string variables by enclosing characters in quotes. Python uses single quotes ' double quotes " and triple quotes """ to denote literal strings. Only the triple quoted strings """ also will automatically continue across the end of line statement.  firstName = 'john'  lastName = "smith"  message = """This is a string that will span across multiple lines. Using newline characters  and no spaces for the next lines. The end of lines within this string also count as a newline when printed""" |  |  |
| c. | Explain while loop in python with example.  The **while loop in Python** is used to iterate over a block of code as long as the test expression (condition) is true. We generally use this **loop** when we don't know the number of times to iterate beforehand.  **Example**: **while loop**  When i is 1, the test expression i <= 5 is true. Hence, the body of the **while loop** is executed. This prints 1 on the screen and the value of i is increased to 2. ... When i is 6, the test expression i <= 5 will be false and the **loop** terminates. |  |  |
| d. | List few in built dictionary functions.   |  |  | | --- | --- | | **Method** | **Description** | | [clear()](https://www.w3schools.com/python/ref_dictionary_clear.asp) | Removes all the elements from the dictionary | | [copy()](https://www.w3schools.com/python/ref_dictionary_copy.asp) | Returns a copy of the dictionary | | [fromkeys()](https://www.w3schools.com/python/ref_dictionary_fromkeys.asp) | Returns a dictionary with the specified keys and value | | [get()](https://www.w3schools.com/python/ref_dictionary_get.asp) | Returns the value of the specified key | | [items()](https://www.w3schools.com/python/ref_dictionary_items.asp) | Returns a list containing a tuple for each key value pair | | [keys()](https://www.w3schools.com/python/ref_dictionary_keys.asp) | Returns a list containing the dictionary's keys | | [pop()](https://www.w3schools.com/python/ref_dictionary_pop.asp) | Removes the element with the specified key | | [popitem()](https://www.w3schools.com/python/ref_dictionary_popitem.asp) | Removes the last inserted key-value pair | | [setdefault()](https://www.w3schools.com/python/ref_dictionary_setdefault.asp) | Returns the value of the specified key. If the key does not exist: insert the key, with the specified value | | [update()](https://www.w3schools.com/python/ref_dictionary_update.asp) | Updates the dictionary with the specified key-value pairs | | [values()](https://www.w3schools.com/python/ref_dictionary_values.asp) | Returns a list of all the values in the dictionary | |  |  |
| e. | What are the different methods to read from a file in python? read() in Python The read() method in Python is a **pre-defined** function which returns the read data in the form of a **string**. The syntax for the read() method is,  **file\_open\_object.read( n )**  Where **file\_open\_object** is the object created while opening a specific file,  and **‘n’** is the number of bytes to be read from the file. In the case where **n** is not specified, the read() function reads the whole file. readline() in Python readline() is yet another pre-defined method in Python, which returns a read line in the form of a **string**. Below is the syntax for readline() function,  **file\_open\_object.readline( n )** readlines() In Python readlines() reads all the **lines** present inside a specified file and returns a list containing the string forms of the read lines. Given below is the syntax,  **file\_open\_object.readlines()**  Using the readlines() method,   |  | | --- | | file=open("new\_file.txt", "r")  print(demo\_file.readlines()) |   **Output:**  ['Python\n', 'C\n', 'C++\n', 'Java\n', 'Kotlin'] |  |  |
| f. | What is constructor in python with example?  The task of **constructors** is to initialize the data members of the class when an object of class is created. The \_\_init\_\_() method is called the **constructor in Python** and is always called when an object is created. class Employee: def \_\_init\_\_(self,name,id): ... emp2 = Employee("David",102 |  |  |
| g. | List few exceptions in Python with an example.  Python Built-in Exceptions   |  |  | | --- | --- | | **Exception** | **Cause of Error** | | EOFError | Raised when the input() function hits end-of-file condition. | | FloatingPointError | Raised when a floating point operation fails. | | GeneratorExit | Raise when a generator's close() method is called. | | ImportError | Raised when the imported module is not found. | |  |  |
| h. | How to read configuration file in database?  Use the **read**() method of SafeConfigParser to **read** the configuration **file**. This program reads the simple. ini **file** from the previous section and prints the value of the url option from the bug\_tracker section. |  |  |
| i. | What is thread synchronization in Python?  **Thread synchronization** is defined as a mechanism which ensures that two or more concurrent **threads** do not simultaneously execute some particular program segment known as critical section. Critical section refers to the parts of the program where the shared resource is accessed. |  |  |
| j. | What is called CGI?  * The Common Gateway Interface, or CGI, is a standard for external gateway programs to interface with information servers such as HTTP servers. * The current version is CGI/1.1 and CGI/1.2 is under progress. |  |  |

**PART – C: (Long Answer Questions) (10Marks)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Answer *ALL* questions | | Marks | [CO#] | [PO#] |
| 3. a. | Explain in detail about two modes of programming in python.   * Interactive mode * Script mode   Interactive mode, also known as the [REPL](https://en.wikipedia.org/wiki/Read%E2%80%93eval%E2%80%93print_loop) provides us with a quick way of running blocks or a single line of Python code. The code executes via the Python shell, which comes with Python installation. Interactive mode is handy when you just want to execute basic Python commands or you are new to Python programming and just want to get your hands dirty with this beautiful language.  The following are the advantages of running your code in interactive mode:   1. Helpful when your script is extremely short and you want immediate results. 2. Faster as you only have to type a command and then press the enter key to get the results. 3. Good for beginners who need to understand Python basics.   **Disadvantages**   1. Editing the code in interactive mode is hard as you have to move back to the previous commands or else you have to rewrite the whole command again. 2. It's very tedious to run long pieces of code.  Script Mode If you need to write a long piece of Python code or your Python script spans multiple files, interactive mode is not recommended. Script mode is the way to go in such cases. In script mode, You write your code in a text file then save it with a .py extension which stands for "Python". Note that you can use any text editor for this, including Sublime, Atom, notepad++, etc.  Advantages   1. It is easy to run large pieces of code. 2. Editing your script is easier in script mode. 3. Good for both beginners and experts.   Disadvantages   1. Can be tedious when you need to run only a single or a few lines of cod. 2. You must create and save a file before executing your code. |  |  |  |
| b. | Write a program to find the ASCII value of the given character (p)..  c = input(“Enter a character”)  print(“ASCII value of “ +c+ “is”+ord(c))  ***Sample Input/Output:***  p  ASCII value of p is 112  *ord()* function convert a character to an integer (ASCII value). It returns the Unicode codepoint of that character |  |  |  |
|  | (OR) |  |  |  |
| c. | Explain in detail about operators.  Operators are the constructs which can manipulate the value of operands.  Consider the expression 4 + 5 = 9. Here, 4 and 5 are called operands and + is called operator.  Types of Operator  Python language supports the following types of operators.   * Arithmetic Operators * Comparison (Relational) Operators * Assignment Operators * Logical Operators * Bitwise Operators * Membership Operators   Identity Operators |  |  |  |
| d. | Write a program to calculate factorial of a given number using function  def factorial(num):  fact=1  i=1  if num==0:  return 1  else:  for i in range(1,num+1):  fact=fact\*i  return fact  no=input(“Enter a number”)  print(factorial(no))  ***Sample input/output:***  Enter a number:5  120 |  |  |  |
|  |  |  |  |  |
| 4. a. | Discuss conditional blocks in detail.   * **Python if Statement Syntax:** * if expression * Statement * else * Statement * **Python if...else Flowchart**     SWITCH CASE STATEMENT  function(argument){  switch(argument) {  case 0:  return "This is Case Zero";  case 1:  return " This is Case One";  case 2:  return " This is Case Two ";  default:  return "nothing";  };  }; The while statement The general syntax for the while statement looks like this:  **while**BOOLEAN\_EXPRESSION:  STATEMENTS  Like the branching statements and the for loop, the while statement is a compound statement consisting of a header and a body. A while loop executes an unknown number of times, as long at the BOOLEAN EXPRESSION is true. |  |  |  |
| b. | Write a program that prints out the first n rows of Pascal’s triangle   |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  | 1 | |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  | 1 | |  | 1 | |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  | 1 | |  | 2 | |  | 1 | |  |  |  | |  |  |  |  | |  |  | |  |  |  |  |  |  | |  | 1 | |  | 3 | |  | 3 | |  |  | 1 | |  | |  |  |  |  |  |  |  |  |  |  |  |  | |  | | 1 | |  | 4 | |  | 6 | |  | 4 | |  | 1 | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |   ***Using Recursion:***  def pascal(n):  if n == 1:  return [ [1] ]  else:  result = pascal(n-1)  lastRow = result[-1]  result.append( [ (a+b) for a,b in zip([0]+lastRow, lastRow+[0]) ] )  return result  def pretty(tree):  if len(tree) == 0: return ‘’  line = ‘ ‘ \* len(tree)  for cell in tree[0]:  line += ‘ %2i’ % cell  return line + “\n” + pretty(tree[1:])  print pretty(pascal(int(6))   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | ***Sample input/output:*** | |  |  |  |  |  | |  |  | 1 |  |  |  |  | |  | 1 | | 1 | |  |  | |  | 1 | 2 |  | 1 |  |  | |  | 1 | 3 |  | 3 | 1 |  | | 1 | 4 | 6 | | 4 | 1 |  | | 1 | 5 | 10 |  | 10 | 5 | 1 |   ***Using iteration:***  def pascal\_triangle(n):  trow = [1]  y = [0]  for x in range(max(n,0)):  print(trow)  trow=[l+r for l,r in zip(trow+y, y+trow)]  return n>=1  pascal\_triangle(6)  ***Sample input/output:***  [1]  [1, 1]  [1, 2, 1]  [1, 3, 3, 1]  [1, 4, 6, 4, 1]  [1, 5, 10, 10, 5, 1] |  |  |  |
|  | (OR) |  |  |  |
| c. | Brief out in-built functions of string, list manipulation and dictionary manipulation. Strings Strings in python are surrounded by either single quotation marks, or double quotation marks.  'hello' is the same as "hello".  You can display a string literal with the print() function: Example print("Hello") print('Hello') Assign String to a Variable Assigning a string to a variable is done with the variable name followed by an equal sign and the string: Example a = "Hello" print(a) Multiline Strings You can assign a multiline string to a variable by using three quotes: Example You can use three double quotes:  a = """Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.""" print(a)  Or three single quotes: Example a = '''Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.''' print(a)  **Note:** in the result, the line breaks are inserted at the same position as in the code. Strings are Arrays Like many other popular programming languages, strings in Python are arrays of bytes representing unicode characters.  However, Python does not have a character data type, a single character is simply a string with a length of 1.  Square brackets can be used to access elements of the string. Example Get the character at position 1 (remember that the first character has the position 0):  a = "Hello, World!" print(a[1]) Looping Through a String Since strings are arrays, we can loop through the characters in a string, with a for loop. Example Loop through the letters in the word "banana":  for x in "banana":   print(x)  **LIST MANIPULATION**  Creation  L =['yellow','red','blue','green','black']  >>>print L  returns:['yellow','red','blue','green','black']  ##########  Accessing / Indexing  L[0]= returns 'yellow'  ##########  Slicing  L[1:4]= returns ['red','blue','green']  L[2:] = returns ['blue','green','black']  L[:2] = returns ['yellow','red']  L[-1] = returns 'black'  L[1:-1]= returns ['red','blue','green']  ##########  Length - number of items inlist  len(L)= returns 5  ##########  Sorting - sorting the list  sorted(L)= returns ['black','blue','green','red','yellow']  ##########  Append - append to end of list  L.append("pink")  >>>print L  returns:['black','blue','green','red','yellow','pink']  ##########  Insert - insert into list  L.insert(0,"white")  >>>print L  returns:['white','black','blue','green','red','yellow','pink']  ##########  Extend - grow list  Delete  Remove an item from a list given its index instead of its value  del.L[0]  >>>print L  ['blue','green','red','yellow','pink']  ##########  Pop  Remove last item in the list  L.pop()= returns 'pink'  # remove indexed value from list  L.pop(1)= returns 'green'  ##########  Reverse - reversing the list  L.reverse()  ##########  Count  Search listandreturn number of instances found  L.count('red')  ##########  Keyword "in"- can be used to test if an item isin a list  if'red'in L:  print"list contains",'red'  ##########  For-in statement - makes it easy to loop over the items in a list  for item in L:  print item  L =['red','blue','green']  for col in L:  print col Dictionary Manipulation Dictionaries are useful whenever you have to items that you wish to link together, and for example storing results for quick lookup.  Create an empty dictionary  months ={}  Create a dictionary with some pairs  # Note: Each key must be unique  months ={1:"January",  2:"February",  3:"March",  4:"April",  5:"May",  6:"June",  7:"July",  8:"August",  9:"September",  10:"October",  11:"November",  12:"December"}  months[1-12] are keys and “January-December” are the values  Print all keys  print"The dictionary contains the following keys: ", months.keys()  Output:  The dictionary contains the following keys: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]  **Accessing**  To get a value out of a dictionary, you must supply its key, you cannot provide the value and get the key  whichMonth = months[1]  print whichMonth  Output: January  To delete an element from a dictionary, use del  del(months[5])  print months.keys()  Output: [1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 12]  To add a new element to a dictionary, assign a value to a new key  months[5]="May"  print months.keys()  Output: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]  To update an element of a dictionary, assign a new value to its key  months[1]="Jan"  print months  Output: {1: ‘Jan’, 2: ‘February’, 3: ‘March’, 4: ‘April’, 5… }  Sorting  sortedkeys = months.keys()  print sortedkeys  Output: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12] |  |  |  |
| d. | Write a program to check whether a number is perfect or not  *Example* : The first perfect number is 6, because 1, 2, and 3 are its proper positive divisors,and 1 + 2 + 3 = 6. Equivalently, the number 6 is equal to half the sum of all its positive divisors: ( 1 + 2 + 3 + 6 ) / 2 = 6.  def perfect\_number(n):# function definition sum = 0  for x in range(1, n):  if n % x == 0:  sum += x  return sum  no=input(“Enter a number”)  sum=perfect\_number(no) # function call  if(sum==no):  print ‘Perfect number’  else:  print ‘Not a Perfect number’  ***Sample input/output:***  Enter a number5  Not a Perfect number  Enter a number6  Perfect number |  |  |  |
|  |  |  |  |  |
| 5. a. | Discuss about file operations.  Access modes govern the type of operations possible in the opened file. It refers to how the file will be used once its opened. These modes also define the location of the **File Handle** in the file. File handle is like a cursor, which defines from where the data has to be read or written in the file. There are 6 access modes in python.   1. **Read Only (‘r’) :**Open text file for reading. The handle is positioned at the beginning of the file. If the file does not exists, raises I/O error. This is also the default mode in which file is opened. 2. **Read and Write (‘r+’) :** Open the file for reading and writing. The handle is positioned at the beginning of the file. Raises I/O error if the file does not exists. 3. **Write Only (‘w’) :** Open the file for writing. For existing file, the data is truncated and over-written. The handle is positioned at the beginning of the file. Creates the file if the file does not exists. 4. **Write and Read (‘w+’)**: Open the file for reading and writing. For existing file, data is truncated and over-written. The handle is positioned at the beginning of the file. 5. **Append Only (‘a’)** : Open the file for writing. The file is created if it does not exist. The handle is positioned at the end of the file. The data being written will be inserted at the end, after the existing data. 6. **Append and Read (‘a+’) :**Open the file for reading and writing. The file is created if it does not exist. The handle is positioned at the end of the file. The data being written will be inserted at the end, after the existing data.   **Python readline()** method will return a line from the [file when called](https://www.edureka.co/blog/file-handling-in-python/).  **readlines() method** will return all the lines in a file in the format of a [list](https://www.edureka.co/blog/lists-in-python/) where each element is a line in the file. |  |  |  |
| b. | Write a program to count the frequency of words in a text file  file=open(“test.txt”,”r+”)  wordcount={} # define a dictionary that holds words its count  for word in file.read().split():# for loop iterates through each word is the file if word not in wordcount:  wordcount[word] = 1  else:  wordcount[word] += 1  for k,v in wordcount.items():  print k, v  file.close()  ***Sample Output:***  This 1  program 2  example 1  Python 2 |  |  |  |
|  | (OR) |  |  |  |
| c. | Discuss about exception handling using python. Handling an exception If you have some *suspicious* code that may raise an exception, you can defend your program by placing the suspicious code in a **try:** block. After the try: block, include an **except:** statement, followed by a block of code which handles the problem as elegantly as possible. Syntax Here is simple syntax of *try....except...else* blocks −  try:  You do your operations here;  ......................  except *ExceptionI*:  If there is ExceptionI, then execute this block.  except *ExceptionII*:  If there is ExceptionII, then execute this block.  ......................  else:  If there is no exception then execute this block.  Here are few important points about the above-mentioned syntax −   * A single try statement can have multiple except statements. This is useful when the try block contains statements that may throw different types of exceptions. * You can also provide a generic except clause, which handles any exception. * After the except clause(s), you can include an else-clause. The code in the else-block executes if the code in the try: block does not raise an exception. * The else-block is a good place for code that does not need the try: block's protection. |  |  |  |
| d. | Write a program to raise an exception when the user input is negative.  try:  a = int(input(“Enter a positive integer value: “))  if a <= 0:  raise ValueError(“This is not a positive number!!”)  except ValueError as ve:  print(ve)  ***Sample input:***  Enter a positive integer value: -1  *Error:*  This is not a positive number!! |  |  |  |
|  |  |  |  |  |
| 6. a. | Explain in detail about SQL database connections using python  Python can be used in database applications.  One of the most popular databases is MySQL.  MySQL Database  To be able to experiment with the code examples in this tutorial, you should have MySQL installed on your computer.  You can download a free MySQL database at <https://www.mysql.com/downloads/>. Install MySQL Driver Python needs a MySQL driver to access the MySQL database.  In this tutorial we will use the driver "MySQL Connector".  We recommend that you use PIP to install "MySQL Connector".  PIP is most likely already installed in your Python environment.  Navigate your command line to the location of PIP, and type the following:  Download and install "MySQL Connector":  C:\Users\Your Name\AppData\Local\Programs\Python\Python36-32\Scripts>python -m pip install mysql-connector-python  Now you have downloaded and installed a MySQL driver. Test MySQL Connector To test if the installation was successful, or if you already have "MySQL Connector" installed, create a Python page with the following content:  demo\_mysql\_test.py:  import mysql.connector  If the above code was executed with no errors, "MySQL Connector" is installed and ready to be used.  Create Connection  Start by creating a connection to the database.  Use the username and password from your MySQL database:  demo\_mysql\_connection.py:  import mysql.connector  mydb = mysql.connector.connect(   host="localhost",   user="yourusername",   password="yourpassword" )  print(mydb) |  |  |  |
| b. | Write a program to create a histogram from a given list of integers.  def histogram( items ):  for n in items:  output = ''  times = n  while( times > 0 ):  output += '\*'  times = times - 1  print(output)  histogram([2, 3, 6, 5])  ***Sample Output:***  \*\*  \*\*\*  \*\*\*\*\*\*  \*\*\*\*\* |  |  |  |
|  | (OR) |  |  |  |
| c. | Briefly explain the process involved in multithreading  In simple words, a **thread** is a sequence of such instructions within a program that can be executed independently of other code. For simplicity, you can assume that a thread is simply a subset of a process!  A thread contains all this information in a **Thread Control Block (TCB)**:   * **Thread Identifier:** Unique id (TID) is assigned to every new thread * **Stack pointer:** Points to thread’s stack in the process. Stack contains the local variables under thread’s scope. * **Program counter:** a register which stores the address of the instruction currently being executed by thread. * **Thread state:** can be running, ready, waiting, start or done. * **Thread’s register set:** registers assigned to thread for computations. * **Parent process Pointer:** A pointer to the Process control block (PCB) of the process that the thread lives on.   Consider the diagram below to understand the relation between process and its thread:    **Multithreading**  Multiple threads can exist within one process where:   * Each thread contains its own **register set** and **local variables (stored in stack)**. * All thread of a process share **global variables (stored in heap)** and the **program code**.   Consider the diagram below to understand how multiple threads exist in memory: |  |  |  |
| d. | Write a program that asks the user to input customer information, call writetofile method to write data to the file and call getall method to retrieve customer information from file  #write data to file  def writetofile(Name,Email=’’,Tel=’’,Address=’’):  try:  f=open(r’customerlist.txt’,’a’)  f.write(Name+’:’+Email+’:’+Tel+’:’+Address+’\n’)  except Exception:’Print error in writing to file...’ finally  f.flush()  f.close()  #Get all customers’information and display  def getall():  f=open(r’customerlist.txt’,’r’)#open file for reading content=f.readlines()#read all lines f.close()  return content  def add():  Name=raw\_input(‘Name:’)  Email=raw\_input(‘Email:’)  Tel=raw\_input(‘Tel:’)  Address=raw\_input(‘Address:’)  writetofile(Name,Email,Tel,Address)  #main program  add()  print getall() |  |  |  |

--- End of Paper ---